

MTR090213

MITRE TECHNICAL REPORT

An Analysis of the CAVES Attestation Protocol using CPSA

December 2009

John D. Ramsdell
Joshua D. Guttman
Jonathan K. Millen
Brian O'Hanlon

Sponsor:	NSA/R23	Contract No.:	W15P7T-08-C-F600
Dept. No.:	G020	Project No.:	0708N6BZ

The views, opinions and/or findings contained in this report are those of The MITRE Corporation and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

Approved for Public Release

© 2009 The MITRE Corporation. All Rights Reserved.

MITRE

Center for Integrated Intelligence Systems
Bedford, Massachusetts

Approved by:

Amy Herzog, 0707N6BZ Project Leader

Abstract

This paper describes the CAVES attestation protocol and presents a tool-supported analysis showing that the runs of the protocol achieve stated goals. The goals are stated formally by annotating the protocol with logical formulas using the rely-guarantee method. The protocol analysis tool used is the Cryptographic Protocol Shape Analyzer.

Contents

1	Introduction	2
1.1	CPSA	2
1.2	Rely-Guarantee Method	3
1.3	Literate Programming	4
2	CAVES Purpose and Features	5
3	The Protocol	6
4	CPSA Overview	11
4.1	Protocols	12
4.2	Executions	13
4.3	Skeletons	14
4.3.1	Preskeletons	15
4.3.2	Shapes	16
4.4	Listeners	16
4.5	Annotations	16
4.6	Run Time Options	18
5	Annotated Roles	18
6	Scenarios	24
7	Trust Argument	28
8	High-Level Attestation Goals	32
9	Protocol Development History	32
10	Conclusion	33
A	Results	34
A.1	Scenario	36
A.2	Scenario	40
A.3	Scenario	43
A.4	Scenario	45
A.5	Scenario	45
A.6	Scenario	46

A.7 Scenario	47
A.8 Scenario	51
A.9 Scenario	52

1 Introduction

This paper describes the CAVES attestation protocol and presents a tool-supported analysis showing that the runs of the protocol achieve stated goals. The protocol analysis tool used is the Cryptographic Protocol Shape Analyzer (CPSA) [4].

An attestation protocol is an exchange of messages over a network by which an appraiser obtains evidence about the state of a target platform. The crucial principles for an attestation architecture, according to [1], are the following, paraphrased for brevity:

Fresh information Evidence should be up to date.

Comprehensive information Evidence should be collected by local measurement tools with access to the entire internal state.

Constrained disclosure A target should be able to identify the appraiser and restrict the evidence sent to it accordingly.

Semantic explicitness The target and type of the evidence should be identified to the appraiser.

Trustworthy mechanism Attestation mechanisms should provide evidence of their trustworthiness.

We shall see in the sections below how CPSA supports verification of attestation protocol properties motivated by these principles. The protocol is only part of the architecture that provides attestation, but it contributes to all of the principles.

1.1 CPSA

To analyze a cryptographic protocol, one finds out what security properties—essentially, authentication and secrecy properties—are true in all its possible executions. The *shapes* of a protocol, relative to some assumptions, are the

minimal, essentially different executions compatible with those assumptions. A protocol may have one, a few, many, or an infinite number of shapes relative to a choice of assumptions. Secrecy and authentication properties can be evaluated by examining the shapes to see if they are all consistent with the desired properties. A “problem statement” means a set of assumptions used to start off a search for shapes. The “scenarios” of Section 6 explore the behavior of CAVES relative to many different problem statements.

A central part of a problem statement is a designation of some long-term keys that are assumed to be uncompromised. These keys are assumed to be used only in accordance with the protocol, i.e. by a principal that executes message transmissions and receptions in order as stipulated by the protocol definition. Another part of a problem statement may be an assumption about some session-specific values, such as session keys or nonces, asserting that these values are created freshly, and will not be re-created independently, whether by an adversary or else by an unlucky protocol participant in a collision of randomly chosen values. A problem statement also conveys some behavior of uncompromised participants, using the uncompromised keys and freshly generated values.

The CPSA program accepts a specification of the protocol and a set of problem statements in an S-expression text format and, upon termination, generates an XHTML document. The output shows the shapes derived for each problem statement. Most protocols and problems yield just a few shapes. If there are very many, or an infinite number, of shapes, CPSA will exit when specified storage bounds are exceeded.

1.2 Rely-Guarantee Method

Some application-specific protocol goals are stated formally by annotating the protocol with logical formulas using the rely-guarantee method [2]. In CPSA, each role of a protocol specifies a sequence of messaging events, each one being either a message transmission or a reception. The rely-guarantee method calls for annotating each event with a formula. The formulas are expressed in a modal logic that allows principals to make local decisions based on assertions made by peers.

The formula annotating a message transmission is a *guarantee* ϕ . The protocol instructs a principal A to ascertain that ϕ is true before sending the message. The formula ϕ may contain parameters that also occur in the message to be sent, and A may fill in these parameters in a specific way

to form a true instance of ϕ . A then transmits the corresponding instance of the message. If A cannot ascertain a true instance of ϕ , then A must not continue this branch of the protocol. If abrupt termination should be avoided, the protocol may provide some error-recovery branch.

The formula annotating a message reception is a *rely* formula. Typically, the rely formula is of the form A says ϕ , or a conjunction of formulas of this form. The rely formula may contain parameters that also appear within the message to be received, so that the message received on a particular occasion determines an instance of the rely formula specified in the protocol definition. The principal B executing the message reception can add this instance of the rely formula to its store of knowledge (its “local theory”). Thus, this new fact may be used later to ascertain the truth of guarantee formulas for future message transmissions.

A protocol is *sound* relative to some assumptions if in every execution compatible with those assumptions, each rely statement is true. For instance, if a particular message transmission B relies on a formula A says ϕ , then there should have been some earlier message transmissions in which the principal A guaranteed formulas ϕ' , such that ϕ is a logical consequence of the formulas ϕ' . That is, A really is committed to the assertion ϕ . The initial local theory of a participant is irrelevant in determining if a protocol is sound.

We call a behavior *regular* if it is an uncompromised local execution that follows the protocol description. We call a principal *regular* if all of its behaviors are regular, normally because its long-term keys have been assumed uncompromised. Only a regular principal can be expected to ascertain the truth of formula before sending a message. The adversary will send messages even when the expected guarantee is false, whenever this appears useful to achieving its malicious goals.

The soundness property allows principals to make local decisions based on guarantees made by other regular principals. Given a set of shapes computed by CPSA for an annotated protocol, another CPSA tool instantiates the formulas; a protocol is sound relative to a set of assumptions if all of the resulting formulas are logically valid.

1.3 Literate Programming

The style of literate programming combines source code and documentation into a single source file. This report is a literate program. This means that in the course of reading this report, you will read the full CPSA specification

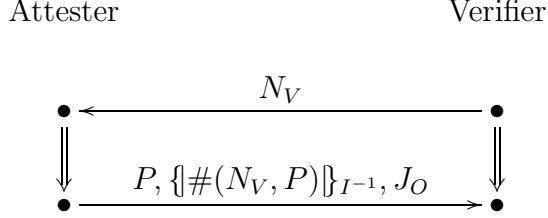


Figure 1: Sailer Protocol

of the CAVES protocol, not just a high-level description of it. The literate programming metalanguage provides a mechanism for presenting specifications or code to the reader in a different order from the way it is supplied to the analysis tool. Thus the protocol can be described in a logical manner.

A reader unfamiliar with CPSA may ignore the fragments of CPSA specifications woven into this document and still understand the conclusions of the paper. A reader familiar with CPSA will see how the tool was used to support the conclusions reached in this paper.

2 CAVES Purpose and Features

A simple protocol for attestation using a TPM was given by Sailer, et al [5]. Their protocol is sketched in Figure 1, using terminology that is closer to our conventions. A Challenger (which we call a Verifier) sends a challenge containing a nonce N_V to a client's Attestation Service (which we call an Attester). The Attester responds with a measurement list J_O and a TPM quote containing PCR values that authenticate the measurements. The quote binds the PCR vector P with a nonce using hashing, and is signed with the identity key I . In this scheme one expects that the measurements are taken once after each system boot, so that the PCR values obtained by extending hash values into the PCRs are predictable.

Confidentiality and authentication for this exchange are provided by embedding it in an SSL session. The actual protocol specifies additional details, such as which measurements and PCRs are requested by the Server. The TNC (Trusted Network Connect) framework devised by the TCG supports this type of attestation.

The CAVES protocol expands upon this basic scheme in several respects.

First, CAVES introduces a Server principal separate from the Verifier. This allows several Servers to share a Verifier, so that fewer Verifiers are needed to keep and maintain a database of acceptable measurements. In effect, the Server is the enforcement point and the Verifier is the decision point.

We have incorporated the certificate handling and session key management into the CAVES protocol, rather than assuming that SSL provides it. This approach gives us flexibility to manage more than the original two principals.

Another feature of CAVES is a provision to request fresh measurements during the attestation session. Depending on PCR usage, it may be impractical to extend hashes of fresh measurements into PCRs, since the result is not predictable unless the PCR can be reset first, or the verifier is able to keep track of the entire history of measurements since the last reboot. In CAVES, these measurements are authenticated cryptographically without using a PCR.

CAVES is intended to be used for a virtualized client architecture with a hypervisor and multiple virtual machines (VMs). One advantage of such an architecture is that a system-dependent or application-dependent measurement agent can be placed in a trusted Measurement VM, isolated from the Client VM containing the user applications and OS. The hypervisor can give the Measurement VM read privileges necessary to examine the Client VM. The Measurement VM is trusted to perform its measurement and bind the measurement result to the TPM report in accordance with the protocol. More generally, one can have an Attestation Manager VM that obtains multiple measurements of (different parts of) the same Client VM from various other measurement agent VMs, and combines the results.

3 The Protocol

The overall plan of CAVES is shown in Figure 3, and the conventions used for variables are in Figure 2. Its roles are Client, Attester, Verifier, Enterprise Privacy Certificate Authority (EPCA), and Server. The name of the protocol was derived from the first letter in each role name. The EPCA role represents the source of identity certificates.

In this protocol, the keys K_S and K_V are public (asymmetric) encryption keys, for which the participants may have PKI certificates. We assume that participants use these PKI certificates to decide whether to regard the

C	c	Client	R	r	Request	N_S	ns	Server Nonce
A	a	Attester	D	d	Data	N_V	nv	Verifier Nonce
V	v	Verifier	M	m	PCR Mask	K	k	Client Key
E	e	EPCA	P	p	PCR Vector	K'	kp	Attester Key
S	s	Server	J	j	Query	I	i	Identity Key
			J_O	jo	Measurement	B	b	Blob

Figure 2: CAVES Legend

matching private parts as uncompromised. The key K is a symmetric session key created by Client C for this interaction with Server S . Key K' is a symmetric session key created by Attester A . Nonces N_S and N_V are generated randomly for this session by the server and the verifier.

Each message description includes a version of the message in CPSA's S-expression syntax. The message S-expression is given as a Nuweb macro definition used to name a fragment or “scrap” of the final specification.

$C \rightarrow S : \{R, A, K\}_{K_S}$: The protocol begins with a request by client C to receive some resource R , which includes a session key K to use for future communication, e.g. to deliver the data D for R . Distinguished name A identifies the identity key to be used by the TPM while gathering evidence about the client.

$\langle \text{initial request 6a} \rangle \equiv$
 $(\text{enc } r \ a \ k \ (\text{pubk } s)) \diamond$

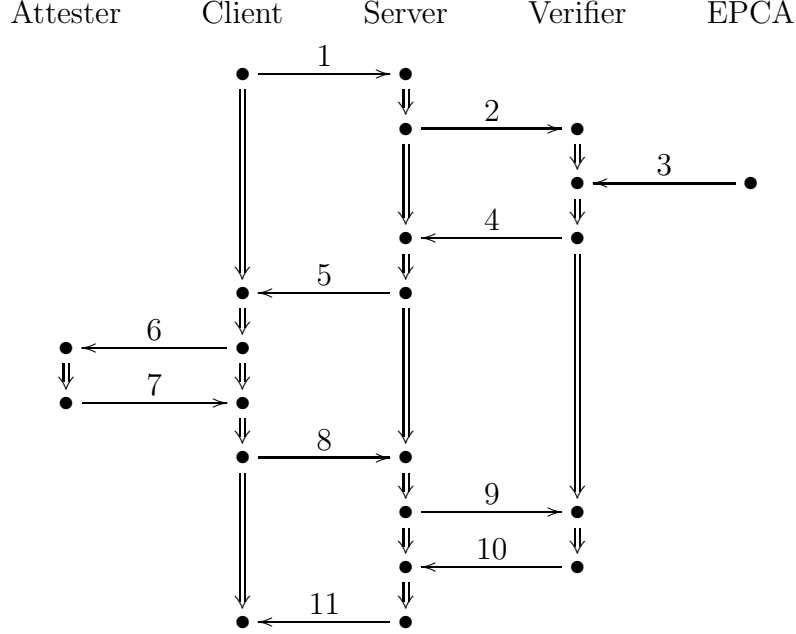
Macro referenced in 19a, 20a.

Nuweb generates the page numbers that follow phrase “Macro referenced in”. A use of a macro includes the page numbers of the scraps that make up its definition.

$S \rightarrow V : \{S, R, A, N_S\}_{K_V}$: Server S delegates to some acceptable verifier V the task of appraising C .

$\langle \text{verification request 6b} \rangle \equiv$
 $(\text{enc } s \ r \ a \ ns \ (\text{pubk } v)) \diamond$

Macro referenced in 20a, 21a.



$$C \rightarrow S : \{R, A, K\}_{K_S} \quad (1)$$

$$S \rightarrow V : \{S, R, A, N_S\}_{K_V} \quad (2)$$

$$E \rightarrow V : \{\text{cert}, A, I, E\}_{K_E^{-1}} \quad (3)$$

$$V \rightarrow S : \{N_V, J, V, N_S, M\}_{K_S} \quad (4)$$

$$S \rightarrow C : \{N_V, J, V, M\}_K \quad (5)$$

$$C \rightarrow A : \{S, N_V, J, V, M, R\}_{\text{ltk}(A,A)} \quad (6)$$

$$A \rightarrow C : \{K', N_V, B\}_{\text{ltk}(A,A)} \quad (7)$$

$$C \rightarrow S : B \quad (8)$$

$$S \rightarrow V : B \quad (9)$$

$$V \rightarrow S : \{\text{valid}, N_S, K'\}_{K_S} \quad (10)$$

$$S \rightarrow C : \{\text{data}, D\}_{K'} \quad (11)$$

$$B = \{K', S, J_O, M, P, \{\#(\#(A, V, R, N_V, J, J_O), M, P)\}_{I^{-1}}\}_{K_V}$$

Figure 3: CAVES Protocol

$E \rightarrow V : \{\{\text{cert}, A, I, E\}\}_{K_E^{-1}} :$ The verifier obtains the identity certificate with A as its distinguished name. The identity key I is certified by an Enterprise Privacy Certification Authority E . For simplicity, the verifier's request for the certificate has been omitted. Perhaps the client sends the certificate with its request for the data.

$\langle \text{identity certificate 8a} \rangle \equiv$
 $(\text{enc "cert" a i e (privk e)}) \diamond$

Macro referenced in 18, 21a.

$V \rightarrow S : \{\{N_V, J, V, N_S, M\}\}_{K_S} :$ The verifier uses A and information in R to select an appropriate query J and a selection mask M , and delivers both to attester A , by forwarding the information through S and C , to C 's local measurement agent.

$\langle \text{verification query 8b} \rangle \equiv$
 $(\text{enc nv j v ns m (pubk s)}) \diamond$

Macro referenced in 20a, 21a.

$S \rightarrow C : \{\{N_V, J, V, M\}\}_K$
 $\langle \text{server query 8c} \rangle \equiv$
 $(\text{enc nv j v m k}) \diamond$

Macro referenced in 19a, 20a.

$C \rightarrow A : \{\{S, N_V, J, V, M, R\}\}_{\text{ltk}(A,A)} :$ The key $\text{ltk}(A, A)$ is not a real key at all. It is an artifact of our method for representing the private inter-VM channel between C and its local attestation service A .

$\langle \text{client query 8d} \rangle \equiv$
 $(\text{enc s nv j v m r (ltk a a)}) \diamond$

Macro referenced in 19a, 22a.

$A \rightarrow C : \{\{K', N_V, B\}\}_{\text{ltk}(A,A)} :$ Attester A retrieves the evidence requested in the form of a large term B which we refer to as the “blob”. As shown

in Fig. 3, the blob is defined by

$$B = \{K', S, J_O, M, P, \{\#(\#(A, V, R, N_V, J, J_O), M, P)\}_{I^{-1}}\}_{K_V}.$$

This message packages J 's output J_O together with the current PCR values P for the registers selected by V in the PCR mask M . This information is generated and authenticated using a TPM quote. The TPM quote uses the hash $\#(A, V, R, N_V, J, J_O)$ of some parameters as a nonce-like seed to be included in a digital signature. The digital signature is prepared using I^{-1} , a TPM-resident Attestation Identity private key.

CPSA has no explicit support for hashing, so a hash is encoded as an asymmetric encryption in which no participant has access to the decryption key. The public key used to create a hash is `hash` and a tag is added to ensure the hash is not confused with other encrypted messages. Thus the hash of variable X , $\#(X)$, is encoded as `(enc "hash" x hash)`.

$$\langle \text{encoded report 9a} \rangle \equiv \\ (\text{enc kp nv } \langle \text{report 9b} \rangle (\text{ltk a a})) \diamond$$

Macro referenced in 22a.

$$\langle \text{report 9b} \rangle \equiv \\ (\text{enc kp s jo m p} \\ (\text{enc} \\ (\text{enc "hash"} \\ (\text{enc "hash" a v r nv j jo hash}) \\ \text{m p hash}) \\ (\text{invk i})) \\ (\text{pubk v})) \diamond$$

Macro referenced in 9a, 21a.

Received message:

$$\langle \text{received report 9c} \rangle \equiv \\ (\text{enc kp nv } \langle \text{blob 10a} \rangle (\text{ltk a a})) \diamond$$

Macro referenced in 19a.

$\langle \text{blob } 10a \rangle \equiv$
 $\text{b} \diamond$

Macro referenced in 9c, 19a, 20a.

$C \rightarrow S : B$

$S \rightarrow V : B$: When the blob is received by C and forwarded through S , C and S treat it as atom B because it is encrypted with V 's public encryption key, to ensure that C and S cannot read it. V uses the identity key in the Attestation Identity Certificate to validate the quote.

$V \rightarrow S : \{\text{valid}, N_S, K'\}_{K_S}$: If the evidence is valid, the server is notified of this fact.

$\langle \text{verification acceptance } 10b \rangle \equiv$
 $(\text{enc "valid" kp ns (pubk s)}) \diamond$

Macro referenced in 20a, 21a.

$S \rightarrow C : \{\text{data}, D\}_{K'}$: The server releases the data to a validated client.

$\langle \text{server response } 10c \rangle \equiv$
 $(\text{enc "data" d kp}) \diamond$

Macro referenced in 19a, 20a.

4 CPSA Overview

An introduction or primer to CPSA is delivered with the program [3]. A brief introduction is included here, along with a description of CPSA's support for the rely-guarantee method. The message terms $\langle term \rangle$ used by CPSA are a straightforward representation of terms using Lisp-style, prefix notation.

A subset of the terms are called *atoms*. Atoms belong to the *base sorts* **name**, **text**, **data**, **skey**, **akey**. Syntactically, atomic terms may be either symbols (i.e., identifiers) or atomic-sorted function applications such as $(\text{pubk } a)$. Even though an atom as a term may have terms within it, a receiver of an atom is not allowed to extract terms that occur in it. This reflects the fact that the reception of the atom $(\text{invk } k)$, the inverse of some asymmetric key k , does not allow the receiver to construct k .

Non-atomic terms are constructed by applications of encryption (**enc**) and pairing (**cat**), where n -ary concatenation is parsed right-associatively. The second argument of an encryption is the key. Encryption may also be written in an n -ary form where the last argument is the key and the arguments preceding it are implicitly concatenated.

A term carries one of its subterms if the possession of the right set of keys allows the extraction of the subterm. The carries relation is the least relation such that (1) t carries t , (2) (**enc** t_0 t_1) carries t if t_0 carries t , and (3) (**cat** t_0 t_1) carries t if t_0 or t_1 carries t . Note that (**enc** t_0 t_1) does not carry t_1 unless (anomalously) t_0 carries t_1 .

4.1 Protocols

A protocol is a set of roles.

(**defprotocol** $\langle sym \rangle$ **basic** $\langle role \rangle^+$)

The symbol $\langle sym \rangle$ names the protocol. The symbol **basic** identifies the term algebra used to specify messages in roles.

A role has the form:

(**defrole** $\langle sym \rangle$ (**vars** $\langle decl \rangle^*$)
 (**trace** $\langle event \rangle^+$)
 (**non-orig** $\langle non \rangle^*$)?
 (**uniq-orig** $\langle atom \rangle^*$)?
 $\langle annos \rangle$)

$\langle non \rangle ::= \langle atom \rangle \mid (\langle height \rangle \langle atom \rangle)$

Non-terminal $\langle sym \rangle$ is an S-expression symbol that names the role. A $\langle decl \rangle$ is a list of variable symbols followed by a sort symbol. The **trace** is a sequence of message events, each indicating a message to be transmitted or received. The syntax used for a message event $\langle event \rangle$ has one of two forms, (**send** $\langle term \rangle$) or (**recv** $\langle term \rangle$). The length of a role is the length of its trace, and must be positive. The remaining components of a role will be described later.

A term originates in a trace if it is carried in some event and the first event in which it is carried is a sending term. A term is acquired by a trace if it first occurs in a receiving term and is also carried by that term.

4.2 Executions

An execution of a protocol may involve any number of strands, each conveying either regular or adversarial behavior. Thus, each strand is an instance of some role. For CPSA input and output, a strand is specified by the following form:

`(defstrand $\langle sym \rangle$ $\langle int \rangle$ $\langle maplet \rangle^*$)`

The symbol names the role, $\langle int \rangle$ is the height which must be positive and no greater than the role's length, and the remainder determines a substitution from role variables to terms.

$\langle maplet \rangle ::= (\langle sym \rangle \langle term \rangle)$

The trace associated with the specified behavior is the result of truncating the role's trace so it agrees with the height, and applying the substitution $(\langle maplet \rangle^*)$.

A strand's behavior includes inherited origination assumptions. When a role assumes atom a is uniquely originating using the **uniq-orig** form, applying the substitution $(\langle maplet \rangle^*)$ to a produces an inherited uniquely originating atom. Role atoms assumed to be non-originating using the **non-orig** form are inherited similarly. For a non-originating assumption, a strand height may be associated with an atom. In this case, a non-originating assumption is inherited by strands that meet or exceed the height constraint. Note that the definition of a uniquely originating atom and a non-originating atom in an execution is still to come.

A strand in an execution is identified by a natural number. To describe an execution, the behavior of each participant is listed sequentially, and position of the **defstrand** form in the list determines the strand's identifier. Zero-based indexing is used, so zero identifies the first strand.

A messaging event in an execution occurs at a node, which is a pair of natural numbers. The first number is the strand's identifier. The second number is the position of an event in the trace of the strand, once again using zero-based indexing. Thus node (1 1) in

```
(defstrand r1 3 (a b) (b a))
(defstrand r2 2 (x a) (y a) (z b))
```

names the last event in the last strand. The term is the result of instantiating the second event in role **r2**'s trace using the substitution $((x\ a)\ (y\ a)\ (z\ b))$.

Message exchanges are part of an execution. Each exchange is described by a pair of nodes. The first node must name a sending term, and the second node must name a receiving term. In an execution, the two terms are the same. Furthermore, for each receiving term in a strand’s trace, there is a unique node that transmits its term. In other words, all message receptions are explained by transmissions within the execution.

In an execution, a *uniquely originating atom* originates in the trace of exactly one strand. An inherited uniquely originating atom must originate in the trace of its strand. CPSA uses uniquely originating atoms to model freshly generated nonces used in many protocols.

A *non-originating atom* is carried by no trace of any strand in an execution, and it or its inverse is the key of an encryption in one of those traces. The inherited non-origination atoms must satisfy this property too.

Strands in executions represent both adversarial and non-adversarial behaviors. A strand that is an instance of a protocol role is non-adversarial, and is called regular. A strand that represents adversarial behavior is called a penetrator strand.

The roles that define adversary behavior codify the basic abilities that make up the Dolev-Yao model. They include transmitting an atom such as a name or a key; transmitting a tag; transmitting an encrypted message after receiving its plain text and the key; and transmitting a plain text after receiving ciphertext and its decryption key. The adversary can also concatenate two messages, or separate the pieces of a concatenated message. Since a penetrator strand that encrypts or decrypts must receive the key as one of its inputs, keys used by the adversary—compromised keys—have always been transmitted by some strand. The basic adversary roles are built into CPSA.

4.3 Skeletons

CPSA never directly represents adversarial behavior. Instead, a skeleton is used. A skeleton represents regular behavior that might make up part of an execution. A skeleton is specified in CPSA output using a `defskeleton` form.

```
(defskeleton <sym> (vars <decl>*)
  <defstrand>+
  (precedes <pair>*)?)
```

$$\begin{aligned}
&(\text{non-orig } \langle atom \rangle^*)^? \\
&(\text{uniq-orig } \langle atom \rangle^*)^?)
\end{aligned}$$

The symbol names the protocol used by its participants. The regular strands are specified as they are in an execution. The precedes form defines a binary relation on nodes ($\langle pair \rangle ::= (\langle node \rangle \langle node \rangle)$). As in an execution, the first node names a sending term and the second term names a receiving term. Unlike an execution, the pair of nodes in the relation need not agree on their message term. Two nodes are related if the sending event precedes the reception event, as an execution it represents may include events in between.

The final two additional components of a skeleton are a set of non-originating atoms, and a set of uniquely originating atoms. To be a skeleton, each uniquely originating atom must originate in at most one strand in the skeleton, and each non-originating atom must never be carried by some event in the skeleton and every variable that occurs in the atom must occur in some event. Furthermore, for each uniquely originating atom that originates in the skeleton, the node relation must ensure that reception nodes that carry the atom follow the node of its origination.

One special skeleton is associated with each execution. It summarizes the regular behavior of the execution. It is derived from the execution by enriching its node relation to contain all node orderings implied by transitive closure, deleting all strands and nodes that refer to penetrator behavior, and then performing the transitive reduction on the resulting node relation. The set of uniquely originating atoms is the set of terms that originate on exactly one strand in the execution, and are carried in a term of a regular strand. The set of non-originating atoms is the union of two sets. One set contains each term that is used as an encryption or decryption key in some term in the execution, but is not carried by any term. The other set contains the terms specified by non-origination assumptions in roles. If a realized skeleton instance maps all of the variables that occur in one of its non-originating role terms, the mapped term is a member of the skeleton's set of non-originating terms. A skeleton is *realized* if it summarizes the behavior of some execution.

4.3.1 Preskeletons

Preskeletons are used to pose problems for CPSA to solve. A preskeleton is similar to a skeleton except atoms assumed to uniquely originate may

originate in more than one strand, and the node relation need not ensure that reception nodes that carry the atom follow some node of origination. Experience has shown that it is more natural to specify some problems in a form that doesn't satisfy all the properties of a skeleton. If CPSA cannot immediately convert its input into a skeleton, an error is signal. With the exception of the restatement of the original problem, all preskeletons printed by CPSA are skeletons. A problem statement is called a *scenario*, and the converted skeleton is called the *scenario skeleton*.

4.3.2 Shapes

Given a scenario skeleton, CPSA determines whether there is an execution containing the strands in the skeleton, and satisfying its origination assumptions. Usually an execution contains additional regular strands, as well as adversary behavior. A major part of what CPSA does is to find all additional regular strands that are necessary to extend the scenario to an execution—a realized skeleton. If a realized skeleton is most-general, in the sense that there is no other realized skeleton that can be instantiated to it by merging nodes or atoms, then it is called a *shape*. CPSA finds all shapes for a scenario.

4.4 Listeners

In addition to the roles specified in a protocol, for each term t , a regular strand may be an instance of a so-called *listener* role with the trace (**recv** t) (**send** t). There are no non-originating or uniquely originating atoms associated with a listener role. Listener behavior is specified with:

(**deflistener** $\langle term \rangle$)

A listener strand is used in a skeleton to assert that a term t is derivable by the adversary, unprotected by encryption. Hence it is used to test for compromise of a term. The term is protected if the resulting skeleton is unrealizable. Otherwise, CPSA will find a shape that shows how the adversary accesses t .

4.5 Annotations

To be analyzed, each role in a protocol must include an **annotations** form, as defined in Table 1. The $\langle term \rangle$ just after the **annotations** symbol is a

$$\begin{aligned}
\langle annos \rangle &::= (\text{annotations } \langle term \rangle ((\langle int \rangle \langle form \rangle)^*)) \\
\langle form \rangle &::= ((\langle sym \rangle \langle fterm \rangle^*) \mid (\text{not } \langle form \rangle)) \\
&\mid (\text{and } \langle form \rangle^*) \mid (\text{or } \langle form \rangle^*) \\
&\mid (\text{implies } \langle form \rangle^* \langle form \rangle) \\
&\mid (\text{iff } \langle form \rangle \langle form \rangle) \\
&\mid (\text{says } \langle term \rangle \langle form \rangle) \\
&\mid (\text{forall } (\langle decl \rangle^*) \langle form \rangle) \\
&\mid (\text{exists } (\langle decl \rangle^*) \langle form \rangle) \\
\langle fterm \rangle &::= \langle term \rangle \mid ((\langle sym \rangle \langle fterm \rangle)^*)
\end{aligned}$$

Table 1: Annotation Syntax

role atom that, when instantiated, is the principal associated with the strand in the shape. A principal may be a key.

What follows is sequences of pairs. The integer gives the position of the event in the trace that is annotated by the formula, using zero-based indexing. Thus, each formula is associated with a node. Nodes for which no formula is specified are implicitly defined to be the trivial formula (**and**) for truth. Use (**or**) for falsehood.

The language of formulas is first-order logic extended with a modal “says” operator. Formula terms may include function symbols that are not part of a protocol’s message signature.

On output, each shape contains an **annotations** form and an **obligations** form. The annotations form presents every non-trivial formula derived from the protocol. The obligations form presents every non-trivial formula that must be true if the shape is sound.

In what follows, annotated roles will be presented in two forms: a tabular form and a CPSA S-expression form. A template for the tabular form follows.

Role 0 (Name P).

$$\begin{array}{ll}
+M_0 & \Phi_0 \\
-M_1 & \Phi_1 \\
+M_2 & \Phi_2
\end{array}$$

A plus sign denotes a sent term, and a minus sign denotes a received term. The S-expression version of the role follows.

(defrole name (vars not specified)

```
(trace (send  $M_0$ ) (recv  $M_1$ ) (send  $M_2$ ))
(annotations  $P$  (0  $\Phi_0$ ) (1  $\Phi_1$ ) (2  $\Phi_2$ )))
```

4.6 Run Time Options

CPSA run time options are specified with a herald form. For this analysis, the bound on the number of strands considered by CPSA has been raised to twelve, and CPSA has been directed to preferentially focus on nonces used by the protocol as opposed to encryptions.

```
"caves.scm" 17  $\equiv$ 
  (herald "CAVES Attestation Protocol"
    (bound 12) (check-nonces)) $\diamond$ 
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

5 Annotated Roles

This section describes the protocol from the perspective of each individual role. The description of each role lists message events and origination assumptions. In support of the rely-guarantee method, a role lists logical formulas that annotate each node. The rely-guarantee annotations will be used in Section 7, and can be ignored on a first pass.

The first role presented is the enterprise privacy certificate authority, due to its simplicity. It transmits the signed attestation identity certificate for distinguished name A , and it guarantees $id(A, I)$, i.e. that A 's identity key is I .

Role 1 (EPCA E).

$$+\{\text{cert}, A, I, E\}_{K_E^{-1}} \quad id(A, I) \tag{12}$$

where I^{-1} is uncompromised. The initial theory of the EPCA defines the id relation.

$\langle \text{enterprise privacy certificate authority role 18} \rangle \equiv$
`(defrole epca (vars (a e name) (i akey))`
`(trace`
`(send ⟨identity certificate 8a⟩))`
`(non-orig (invk i))`
`(annotations e`
`(0 (id a i))))◇`

Macro referenced in 23.

The client requests data in Message 13 and receives the data in Message 18. The other messages relay information between the server and the attester. In this protocol, no participant takes note of the name associated the client, the variable C . The variable C appears in Message 13 as an artifact of our implementation—the rely-guarantee analysis software requires that every role declare a principal. The sole purpose for transmitting C is to satisfy this requirement. Because no other role makes use of C , its presence does not affect the shapes produced by CPSA.

Role 2 (Client C).

$$+(C, \{R, A, K\}_{K_S}) \quad (13)$$

$$-\{N_V, J, V, M\}_K \quad (14)$$

$$+\{S, N_V, J, V, M, R\}_{\text{ltk}(A,A)} \quad (15)$$

$$-\{K', N_V, B\}_{\text{ltk}(A,A)} \quad (16)$$

$$+B \quad (17)$$

$$-\{\text{data}, D\}_{K'} \quad S \text{ says } \text{resource}(R, D) \quad (18)$$

where K is fresh.

$\langle \text{client role 19a} \rangle \equiv$
`(defrole client (vars $\langle \text{client declarations 19b} \rangle$)`
`(trace`
`(send (cat c $\langle \text{initial request 6a} \rangle$)))`
`(recv $\langle \text{server query 8c} \rangle$)`
`(send $\langle \text{client query 8d} \rangle$)`
`(recv $\langle \text{received report 9c} \rangle$)`
`(send $\langle \text{blob 10a} \rangle$)`
`(recv $\langle \text{server response 10c} \rangle$))`
`(uniq-orig k)`
`(annotations c`
`(5 (says s (resource r d))))))` \diamond

Macro referenced in 23.

$\langle \text{client declarations 19b} \rangle \equiv$
`(c a v s name) (r m j d text)`
`(nv data) (k kp skey) (b mesg)` \diamond

Macro referenced in 19a.

Role 3 (Server S).

$$-\{R, A, K\}_{K_S} \quad (19)$$

$$+\{S, R, A, N_S\}_{K_V} \quad \text{verifier}(V) \quad (20)$$

$$-\{N_V, J, V, N_S, M\}_{K_S} \quad (21)$$

$$+\{N_V, J, V, M\}_K \quad (22)$$

$$-B \quad (23)$$

$$+B \quad (24)$$

$$-\{\text{valid}, N_S, K'\}_{K_S} \quad V \text{ says } \text{approved}(R, A, N_V) \quad (25)$$

$$+\{\text{data}, D\}_{K'} \quad \text{approved}(R, A, N_V) \wedge \text{resource}(R, D) \quad (26)$$

where N_S is fresh. The initial theory of the server defines the *verifier* and *resource* relations and contains the following rule.

$$\text{approved}(r, a, n) \leftarrow \text{verifier}(v) \wedge v \text{ says } \text{approved}(r, a, n) \quad (27)$$

$\langle \text{server role 20a} \rangle \equiv$
 $(\text{defrole server (vars } \langle \text{server declarations 20b} \rangle)$
 $(\text{trace}$
 $(\text{recv } \langle \text{initial request 6a} \rangle)$
 $(\text{send } \langle \text{verification request 6b} \rangle)$
 $(\text{recv } \langle \text{verification query 8b} \rangle)$
 $(\text{send } \langle \text{server query 8c} \rangle)$
 $(\text{recv } \langle \text{blob 10a} \rangle)$
 $(\text{send } \langle \text{blob 10a} \rangle)$
 $(\text{recv } \langle \text{verification acceptance 10b} \rangle)$
 $(\text{send } \langle \text{server response 10c} \rangle))$
 (uniq-orig ns)
 $(\text{annotations s}$
 $(1 (\text{verifier v}))$
 $(6 (\text{says v (approved r a nv)}))$
 $(7 (\text{and (approved r a nv) (resource r d)})))) \diamond$

Macro referenced in 23.

$\langle \text{server declarations 20b} \rangle \equiv$
 $(\text{a v s name}) (\text{r m j d text})$
 $(\text{ns nv data}) (\text{k kp skey}) (\text{b mesg}) \diamond$

Macro referenced in 20a.

Role 4 (Verifier V).

$$-\{S, R, A, N_S\}_{K_V} \quad (28)$$

$$-\{\text{cert}, A, I, E\}_{K_E^{-1}} \quad E \text{ says } id(A, I) \quad (29)$$

$$+\{N_V, J, V, N_S, M\}_{K_S} \quad ask(R, A, J, M) \quad (30)$$

$$-B \quad A \text{ says } meas(I, N_V, J, J_O, M, P) \quad (31)$$

$$+\{\text{valid}, N_S, K'\}_{K_S} \quad approved(R, A, N_V) \quad (32)$$

where $B = \{K', S, J_O, M, P, \{\#(\#(A, V, R, N_V, J, J_O), M, P)\}_{I^{-1}}\}_{K_V}$, N_V is fresh and K_P^{-1} is uncompromised. The initial theory of the verifier defines

the *epca* and *ok* relations, and contains the following rules.

$$ask(r, a, j, m) \leftarrow ok(r, a, j, j_o, m, p) \quad (33)$$

$$id(a, i) \leftarrow epca(e) \wedge e \text{ says } id(a, i) \quad (34)$$

$$approved(r, a, n) \leftarrow id(a, i) \wedge$$

$$a \text{ says } meas(i, n, j, j_o, m, p) \wedge ok(r, a, j, j_o, m, p)$$

$\langle \text{verifier role 21a} \rangle \equiv$

```
(defrole verifier (vars  $\langle$  verifier declarations 21b $\rangle$ )
  (trace
    (recv  $\langle$  verification request 6b $\rangle$ )
    (recv  $\langle$  identity certificate 8a $\rangle$ )
    (send  $\langle$  verification query 8b $\rangle$ )
    (recv  $\langle$  report 9b $\rangle$ )
    (send  $\langle$  verification acceptance 10b $\rangle$ ))
  (non-orig (invk hash) (privk e) (5 (ltk a a)))
  (uniq-orig nv)
  (annotations v
    (1 (says e (id a i)))
    (2 (ask r a j m))
    (3 (says a (meas i nv j jo m p)))
    (4 (approved r a nv)))) $\diamond$ 
```

Macro referenced in 23.

$\langle \text{verifier declarations 21b} \rangle \equiv$

```
(a v e s name) (r m p j jo text)
(ns nv data) (hash i akey) (kp skey) $\diamond$ 
```

Macro referenced in 21a.

The channel key $ltk(A, A)$ may be assumed to be non-originating or not, depending on whether we wish to assume that both of the VMs that communicate over this channel are uncompromised. If $ltk(A, A)$ is non-originating, then two conclusions follow. First, the contents of communications along this channel are not disclosed. Second, the endpoints of the channel, C and A , are using the channel only in accordance with this protocol. Thus, any messages sent or received over this channel are part of a regular, protocol-respecting, execution of one of the roles.

If $\text{ltk}(A, A)$ is not assumed to be non-originating, that means that information on this channel is available to an adversary, which could happen if one or both of the endpoints are compromised.

An instance of a verifier role of height five has made a trust decision—that the supplied measurement is acceptable. The decision is reflected by the non-origination assumption (5 (1tk a a)) in the role asserting that the channel key $\text{ltk}(A, A)$ is uncompromised.

Role 5 (Attester A).

$$-\{S, N_V, J, V, M, R\}_{\text{ltk}(A, A)} \quad (36)$$

$$+B \quad \text{verifier}(V) \wedge \text{meas}(I, N_V, J, J_O, M, P) \quad (37)$$

where $B = \{K', S, J_O, M, P, \{\#(\#(A, V, R, N_V, J, J_O), M, P)\}_{I^{-1}}\}_{K_V}$. The initial theory of the attester defines the *verifier* relation.

```

⟨ attester role 22a ⟩ ≡
  (defrole attester (vars ⟨ attester declarations 22b ⟩)
    (trace
      (recv ⟨ client query 8d ⟩)
      (send ⟨ encoded report 9a ⟩))
    (uniq-orig kp)
    (non-orig (invk hash))
    (annotations a
      (1 (and (verifier v) (meas i nv j jo m p))))))◇

```

Macro referenced in 23.

```

⟨ attester declarations 22b ⟩ ≡
  (a v s name) (r m p j jo text)
  (nv data) (hash i akey) (kp skey)◇

```

Macro referenced in 22a.

The CAVES protocol contains the five roles.

```
"caves.scm" 23 ≡
  (defprotocol caves basic
    ⟨ attester role 22a ⟩
    ⟨ client role 19a ⟩
    ⟨ server role 20a ⟩
    ⟨ verifier role 21a ⟩
    ⟨ enterprise privacy certificate authority role 18 ⟩)◇
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

6 Scenarios

Most of the scenarios investigate the properties of the protocol as seen from the point of view of one of the participants. The scenario usually contains a regular strand for just one of the roles. We can then ask an authentication question: if this strand completes normally, are the other roles identified in this strand actually present in the execution, and, if so, do they agree on significant data such as keys and the identity of other parties?

We can also ask a confidentiality question for data that is uniquely originated in the scenario role. A confidentiality question is tested by adding a listener strand for the data in question. Most of the attestation protocol principles are addressed by authentication and confidentiality questions, and others are addressed by annotations.

The “bottom line” of the protocol is the delivery of the data d to the client. The confidentiality of d is an essential goal, since the point of attestation is to ensure that the data goes only to a client with acceptable measurements.

Origination assumptions play an important part in setting up and interpreting scenarios. For the CAVES protocol we pay special attention to the origination assumption regarding the channel key $\text{ltk}(A, A)$. The channel key $\text{ltk}(A, A)$ may be assumed to be non-originating or not, depending on whether we wish to assume that both of the VMs (the client and the attester) that communicate over this channel are uncompromised. If $\text{ltk}(A, A)$ is non-originating, then two conclusions follow. First, the contents of communications along this channel are not disclosed. Second, the endpoints of the channel, C and A , which use this channel, act only in accordance with this protocol. Thus, any messages sent or received over this channel are part of a regular, protocol-respecting, execution of one of the roles.

If $\text{ltk}(A, A)$ is not assumed to be non-originating, that means that infor-

mation on this channel may be available to an adversary. If one or both of the endpoints are compromised, or if the hypervisor exposes the channel contents, this would occur.

The first scenario analyzes the protocol under the assumption of a complete run of the verifier. Since the verifier approved the release of the data, the measurements reported by the attester are acceptable, and the verifier can justify the assumption that the attester and its associated client are regular. This is indicated by assuming that $(\text{ltk } a \ a)$ is non-originating by assuming the verifier has height five. In the CPSA results, we wish to confirm an authentication goal, namely, that there was a regular attester strand agreeing on the name a of the attester principal, and agreeing on the measurements p and jo used to verify acceptability.

Scenario 1 (Full Length Verifier).

"caves.scm" 24 \equiv

```
(defskeleton caves
  (vars (a v e s name))
  (defstrand verifier 5 (a a) (v v) (e e) (s s))
  (non-orig (privk v) (privk e) (privk s))) $\diamond$ 
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

CPSA finds a shape with the normal five strands, where the height of each strand is the full length with the exception of the client (height 5) and the server (height 4), since the verifier has no information about what happened after its own last message. The values on which the strands agree is as expected.

The fresh information principle is respected in this scenario because the verifier generated the nonce nv before the measurement blob was created.

To illustrate the importance of the non-origination assumption on $(\text{ltk } a \ a)$ that was added because the verifier had height five, a similar scenario is run where the verifier has height four.

Scenario 2 (Verifier Before Decision).

"caves.scm" 25a \equiv

```
(defskeleton caves
  (vars (e s name))
  (defstrand verifier 4 (e e) (s s))
  (non-orig (privk e) (privk s))) $\diamond$ 
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

CPSA finds a shape with four strands—it cannot infer the existence of a client strand. Thus, the session key k received by the server may have been generated by the adversary.

The following scenarios are from the point of view of the attester. First, suppose that $(ltk\ a\ a)$ is non-originating.

Scenario 3 (Full Length Attester).

"caves.scm" 25b \equiv

```
(defskeleton caves
  (vars (a v name))
  (defstrand attester 2 (a a) (v v))
  (non-orig (privk v) (ltk a a))) $\diamond$ 
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

CPSA finds a shape with the attester and a regular client, but nothing else. Now, what if $(ltk\ a\ a)$ is not non-originating?

Scenario 4 (Full Length Attester, Compromised Client).

"caves.scm" 25c \equiv

```
(defskeleton caves
  (vars (a v name))
  (defstrand attester 2 (a a) (v v))
  (non-orig (privk v))) $\diamond$ 
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

In this case, CPSA finds a shape with the attester only. There is still a client being measured, but the client may not be regular.

One of the attestation principles is constrained disclosure. We can test this by adding a listener for the measurements jo or p . The following is the test for jo . Note that jo must be assumed uniquely originating in order for the confidentiality test with the listener to be meaningful.

Scenario 5 (Full Length Attester, Compromised Client, Listener for jo).
"caves.scm" 26a \equiv

```
(defskeleton caves
  (vars (a v name) (jo text))
  (defstrand attester 2 (a a) (v v) (jo jo))
  (deflistener jo)
  (uniq-orig jo)
  (non-orig (privk v)))◇
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

CPSA does not find a realized shape with the listener, confirming that the measurement is kept confidential. A similar test for p shows that the PCR vector is kept confidential as well.

Scenario 6 (Full Length Attester, Compromised Client, Listener for p).
"caves.scm" 26b \equiv

```
(defskeleton caves
  (vars (a v name) (p text))
  (defstrand attester 2 (a a) (v v) (p p))
  (deflistener p)
  (uniq-orig p)
  (non-orig (privk v)))◇
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

The next scenario, with a full-length server, should yield a normal execution of the protocol.

Scenario 7 (Full Length Server).
"caves.scm" 26c \equiv

```
(defskeleton caves
  (vars (v s name))
  (defstrand server 8 (s s) (v v))
  (non-orig (privk s) (privk v)))◇
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

CPSA finds the expected execution.

To this scenario, we add a listener for d to check that the data is not compromised.

Scenario 8 (*D* Kept Secret in Full Length Server).

"caves.scm" 27a \equiv

```
(defskeleton caves
  (vars (v s name) (d text))
  (defstrand server 8 (s s) (v v) (d d))
  (deflistener d)
  (uniq-orig d)
  (non-orig (privk s) (privk v))) $\diamond$ 
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

CPSA finds no shape, so the secret is kept.

The client has an authentication goal to ensure that the data it receives comes from the chosen server. Since the client is regular, we assume a safe channel key, and since the client selects the server, we assume the server's public key is safe as well.

Scenario 9 (Full Length Client).

"caves.scm" 27b \equiv

```
(defskeleton caves
  (vars (a v s name) (k skey))
  (defstrand client 6 (v v) (a a) (s s) (k k))
  (non-orig (privk s) (privk v) (ltk a a))) $\diamond$ 
```

File defined by 17, 23, 24, 25abc, 26abc, 27ab.

CPSA produces a normal shape as in Scenario 7 In particular, the client and server agree on their identities and on the data sent to the client.

7 Trust Argument

Figure 4 shows the rely and guarantee formulas together as specified in Section 5. These formulas also appear in the CPSA output for Scenario 7, where the formulas are instantiated in a way that is consistent across strands.

The purpose of rely and guarantee formulas is to supply application-specific context for the data sent in messages. When the server chooses a verifier to contact, for example, the guarantee of *verifier*(*V*) is what identifies *V* as the name of a principal authorized to play a verifier role. The implementor or administrator of the protocol has the responsibility to ensure in

some way, not specified in the protocol itself, that only appropriate verifier principals are chosen. Similar remarks apply to other guarantees, such as those that choose or evaluate measurements for acceptability.

For each non-trivial formula that annotates the shape, Table 4 contains a row. The row entries are (1) the node of the formula, (2) the principal associated with its strand, and (3) the formula. A node with a plus sign is a transmission node and its formula is guaranteed before the message is sent. A node with a minus sign is a reception node, and when the shape is sound, its formula can be relied upon as a result of receiving the message. CPSA does not itself check soundness, but it does generate proof obligations stating that a rely formula ψ follows from the formulas $A \text{ says } \phi$, where each formula ϕ was guaranteed by principal A at some transmission node that precedes ψ in the node ordering. An external theorem prover could be used to check that each such formula $(\bigwedge A \text{ says } \phi) \supset \psi$ is logically valid.

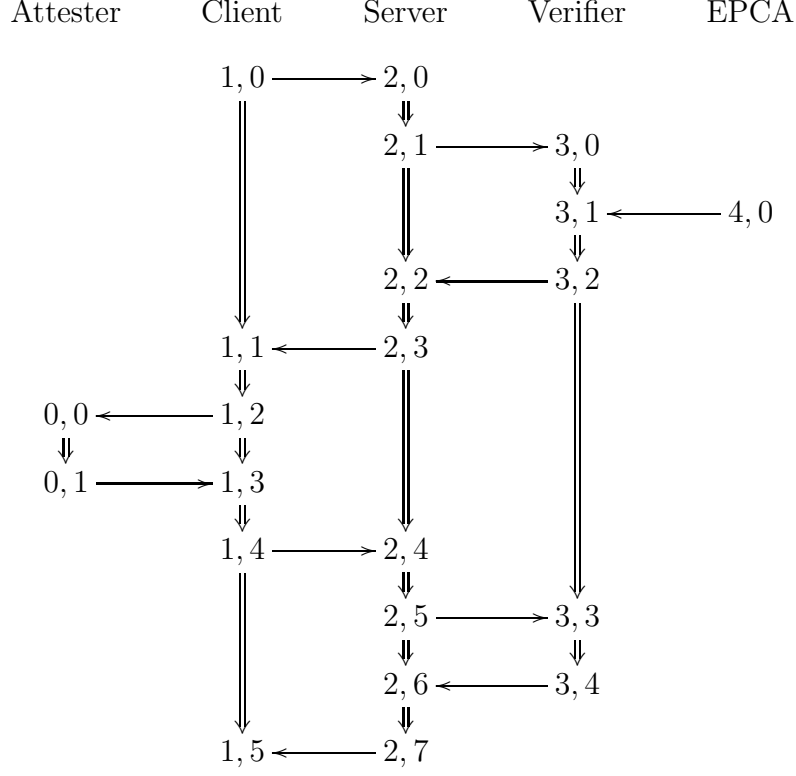
In all shapes found for CAVES, the soundness proof obligations are trivially true. For example, the guarantee made by the attester at node $(0, 1)$ justifies the reliance by the verifier on the formula at node $(3, 3)$.

From the client’s point of view, the client wants to know that the received data was sent by the specified server for its resource request. Scenario 9 shows that a regular server exists and agrees with the client on their identities and the parameter D , as well as the request R , but the client depends on the rely formula at node $(1, 5)$ in Figure 4 to decide that D is the requested data.

A goal of the protocol from the server’s point of view is that only a valid client receives the resource data. There are three questions buried in this concern. One is that the client is “valid”. Another is that only the identified client receives the data. And, finally, that content D of the message is in fact the requested resource data.

We have already tested, in Scenario 8, that D is not compromised by the adversary, so it is received only by a regular participant. But is that participant “valid”? And, if it is, we must still ask why we think that D is the resource data. The answer to the latter question is the guarantee *resource*(R, D) in node $+(2, 7)$. Of course, only the implementor knows what kind of database operation or computation was called for to produce that data.

Why is the recipient of the data “valid”? In this case, “valid” is defined by the guarantee *approved*(R, A, N_V) at node $+(2, 7)$. This guarantee is justified by Inference Rule 27 on Page 20 which depends on hypotheses provided by annotations at nodes $+(2, 1)$ and $-(2, 6)$.



- + (2, 1) *S* *verifier(V)*
- + (4, 0) *E* *id(A, I)*
- (3, 1) *V* *E* says *id(A, I)*
- + (3, 2) *V* *ask(R, A, J, M)*
- + (0, 1) *A* *meas(I, N_V, J, J_O, M, P)*
- (3, 3) *V* *A* says *meas(I, N_V, J, J_O, M, P)*
- + (3, 4) *V* *approved(R, A, N_V)*
- (2, 6) *S* *V* says *approved(R, A, N_V)*
- + (2, 7) *S* *resource(R, D) ∧ approved(R, A, N_V)*
- (1, 5) *C* *S* says *resource(R, D)*

Figure 4: CAVE Rely-Guarantee Formulas

The validity of guarantees justified by inference rules depends on the validity of the inference rules and the chain of deductions using them. This process is not checked by CPSA, but the necessary formulas and rules are made visible, and other tools can be brought to bear to check them.

The rest of the argument is summarized here to suggest why several of the other rules and guarantees are needed. Most of the logical activity occurs in the verifier.

The verifier guarantees the client associated with verification session N_V is valid before sending the message at node (3,4) using Inference Rule 35 on Page 22. To use this rule, the verifier must obtain the attestation identity key of the attester, receive a measurement from the attester, and check that the measurement values are correct by consulting the *ok* relation in its initial theory.

At node (3,0), the verifier receives the distinguished name A used in the attestation identity certificate containing the attester's identity key. After receiving the certificate at node (3,1), the verifier relies on the fact that the enterprise privacy certificate authority E says that I is A 's identity key. After consulting the *epca* relation in its initial theory, and using the Inference Rule 34, the verifier guarantees that I is A 's identity key at node (3,4).

The verifier relies on the fact that attester A claims it provides some measurements of the client after receiving the message at node (3,3). For this shape, the measured values agree with the expected values, otherwise, the message transmission at node (3,4) would have been forbidden because its formula could not be guaranteed.

The guarantee at node (3,2) and Inference Rule 33 select a PCR mask M and a measurement query J that allows the check of the measurements before node (3,4) using Inference Rule 35.

The last non-trivial formula to be discussed is the guarantee made by the attester. It asserts that V is a verifier, and for verification session N_V , the requested measurement result is J_O for query J , and the requested PCR vector is P for PCR mask M . It must guarantee that V is verifier so as to ensure the measurements are available only to a party authorized to view them.

The server delegates to the verifier the decision to release data to the client. The verifier approves the release by sending its last message only if the attester provides credible evidence that the client is valid. The rely-guarantee formulas formalize this trust argument.

8 High-Level Attestation Goals

Fresh information Evidence is up to date because the attester’s nodes are between the verifier nodes (3, 2) and (3, 3). (See e.g. Scenario 1.)

Comprehensive information The query J may be chosen to provide sufficient insight into the state of the client C , as reported in output J_O , to make an adjudication that C will behave regularly.

Constrained disclosure The measurement is not available to the client and the server, and the attester authenticates the verifier before sending out the measurement. (See e.g. Scenario 5.)

Semantic explicitness The use of the rely-guarantee method provides formal description of the access decision.

Trustworthy mechanism The TPM provides a root of trust for reporting via the TPM quote included in the message sent at node (0, 1).

9 Protocol Development History

An earlier version of this protocol omitted S and N_V from Message 6 in Figure 3 and the contents of the blob B . A CPSA analysis of this version revealed a man-in-the-middle attack due to a possible disagreement on the identity of the client between the server and the attester. Some protocol modifications were tried, and several CPSA runs were made. For a sample of the analysis, consider the following result of a run in which the only difference from the current protocol is the absence of S from the attester messages. The scenario had a full-length server and a listener for D , and the channel key was assumed non-originating. This run yielded a shape with all roles represented, and with the following strand mappings:

```
(defstrand server 8 (b b) (r r) (m m) (j j) (d d) (ns ns)
  (nv nv) (a a) (v v) (s s) (k k))
(deflistener d)
(defstrand epca 1 (a a) (e e) (i i))
(defstrand verifier 5 (r r) (m m) (p p) (j j) (jo jo) (ns ns)
  (nv nv) (a a) (v v) (e e) (s s) (hash hash) (i i))
(defstrand attester 2 (r r) (m m) (p p) (j j) (jo jo) (nv nv))
```

```

(a a) (v v) (hash hash) (i i))
(defstrand client 3 (r r) (m m) (j j) (nv nv) (s s-0) (c c)
(a a) (v v) (k k-0))

```

The anomaly here is that the client disagrees with the server on the server identity $s-0$ and the session key $k-0$. The data d was compromised because the server received a different session key k from the adversary. For the attack to work, the adversary must possess the private key of $s-0$. That is, the client is communicating with a malicious server.

Omitting the occurrence of N_V outside the blob in Message 7 in Figure 3 allows shapes with two clients. Inserting N_V outside the blob was the final protocol design step.

10 Conclusion

In this report, we have highlighted the role that CPSA analysis has played in refining and justifying the design of the CAVES protocol. Shapes analysis without annotations allowed us to prove authentication and confidentiality properties. The rely and guarantee annotations bridge the gap between message behavior and its application-specific semantics. CPSA also allowed us to check the soundness of the reliance by one participant on another participant’s guarantees.

The process of representing the protocol in the CPSA language for purposes of analysis led to formulation of generally applicable techniques for modeling such phenomena as private channels—using an auxiliary long-term key like $\text{ltk}(A, A)$ —and hash functions.

We hope that the use of Nuweb to weave together the specification text and the associated exposition has demonstrated how we can effectively combine versatility of narrative style with a tight coupling to the actual executable text.

This analysis of CAVES is one step in a larger plan to analyze related protocols such as the remeasurement protocol and a more general protocol that can negotiate and launch a selection of measurement and attestation functions. We also plan to address other questions associated with logical annotations, such as how inference rules may be justified by analysis based on models of the TRP itself and its components.

References

- [1] George Coker, Joshua Guttman, Peter Loscocco, Justin Sheehy, and Brian Sniffen. Attestation: Evidence and trust. In *ICICS '08: Proceedings of the 10th International Conference on Information and Communications Security*, pages 1–18, Berlin, Heidelberg, 2008. Springer-Verlag.
- [2] Joshua D. Guttman, Jonathan C. Herzog, F. Javier Thayer, Jay A. Carlson, John D. Ramsdell, and Brian T. Sniffen. Trust management in strand spaces: A rely-guarantee method. In *LNCS*, volume 3705, pages 116–145. Springer-Verlag, 2005.
- [3] John D. Ramsdell and Joshua D. Guttman. *CPSA Primer*. The MITRE Corporation, Bedford, MA, USA, 1.4 edition, 2009. Delivered with the CPSA program.
- [4] John D. Ramsdell and Joshua D. Guttman. CPSA: A cryptographic protocol shapes analyzer. In *Hackage*. 2010. <http://hackage.haskell.org/package/cpsa>.
- [5] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

A Results

What follows is the verbatim output of CPSA except that duplicate protocols listings have been removed and scenarios section titles added. If available, the XHTML rendering of this information should be preferred as hyperlinks and SVG diagrams enhance the presentation.

```
(comment "CPSA 2.2.0")
(comment "Annotated skeletons")
(comment "CPSA 2.2.0")
(comment "Extracted shapes")
(herald "CAVES Attestation Protocol" (bound 12) (check-nonces))
(comment "CPSA 2.2.0")
(comment "All input read")
```

```

(comment "Strand count bounded at 12")
(comment "Nonces checked first")

(defprotocol caves basic
  (defrole attester
    (vars (a v s name) (r m p j jo text) (nv data) (hash i akey)
          (kp skey))
    (trace (recv (enc s nv j v m r (ltk a a)))
          (send
            (enc kp nv
              (enc kp s jo m p
                (enc
                  (enc "hash" (enc "hash" a v r nv j jo hash) m p
                    hash) (invk i)) (pubk v)) (ltk a a))))
          (non-orig (invk hash))
          (uniq-orig kp)
          (annotations a (1 (and (verifier v) (meas i nv j jo m p))))))
    (defrole client
      (vars (c a v s name) (r m j d text) (nv data) (k kp skey)
            (b mesg))
      (trace (send (cat c (enc r a k (pubk s))))
            (recv (enc nv j v m k)) (send (enc s nv j v m r (ltk a a)))
            (recv (enc kp nv b (ltk a a))) (send b)
            (recv (enc "data" d kp)))
            (uniq-orig k)
            (annotations c (5 (says s (resource r d)))))
      (defrole server
        (vars (a v s name) (r m j d text) (ns nv data) (k kp skey)
              (b mesg))
        (trace (recv (enc r a k (pubk s)))
              (send (enc s r a ns (pubk v)))
              (recv (enc nv j v ns m (pubk s))) (send (enc nv j v m k))
              (recv b) (send b) (recv (enc "valid" kp ns (pubk s)))
              (send (enc "data" d kp)))
              (uniq-orig ns)
              (annotations s (1 (verifier v)) (6 (says v (approved r a nv)))
                (7 (and (approved r a nv) (resource r d)))))
        (defrole verifier
          (vars (a v e s name) (r m p j jo text) (ns nv data)
                (hash i akey) (kp skey))

```

```

(trace (recv (enc s r a ns (pubk v)))
  (recv (enc "cert" a i e (privk e)))
  (send (enc nv j v ns m (pubk s)))
  (recv
    (enc kp s jo m p
      (enc
        (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
        (invk i)) (pubk v)))
    (send (enc "valid" kp ns (pubk s))))
  (non-orig (invk hash) (privk e) (5 (ltk a a)))
  (uniq-orig nv)
  (annotations v (1 (says e (id a i))) (2 (ask r a j m))
    (3 (says a (meas i nv j jo m p))) (4 (approved r a nv))))
(defrole epca
  (vars (a e name) (i akey))
  (trace (send (enc "cert" a i e (privk e))))
  (non-orig (invk i))
  (annotations e (0 (id a i))))

```

A.1 Scenario

verifier



```

(defskeleton caves
  (vars (r m p j jo text) (ns nv data) (a v e s name) (kp skey)
    (hash i akey))
  (defstrand verifier 5 (r r) (m m) (p p) (j j) (jo jo) (ns ns)

```

```

(nv nv) (a a) (v v) (e e) (s s) (kp kp) (hash hash) (i i))
(non-orig (ltk a a) (invk hash) (privk v) (privk e) (privk s))
(uniq-orig nv)
(traces
  ((recv (enc s r a ns (pubk v)))
    (recv (enc "cert" a i e (privk e)))
    (send (enc nv j v ns m (pubk s)))
    (recv
      (enc kp s jo m p
        (enc
          (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
          (invk i)) (pubk v)))
      (send (enc "valid" kp ns (pubk s)))))
(label 0)
(unrealized (0 1) (0 3))
(comment "1 in cohort - 1 not yet seen"))

```


38

```

      (invk i)) (pubk v))) (r r) (m m) (j j) (nv nv) (c c)
    (a a) (v v) (s s) (k k) (kp kp))
  (precedes ((0 2) (2 2)) ((1 0) (0 1)) ((2 1) (0 0))
    ((2 3) (4 1)) ((3 1) (4 3)) ((4 0) (2 0)) ((4 2) (3 0))
    ((4 4) (0 3)))
  (non-orig (ltk a a) (invk hash) (invk i) (privk v) (privk e)
    (privk s))
  (uniq-orig ns nv k kp)
  (operation nonce-test
    (contracted (r-0 r) (s-0 s) (c-0 c) (v-0 v) (m-0 m) (j-0 j)
      (k-0 k)) nv (5 1) (enc nv j v m k)
    (enc nv j v ns m (pubk s)))
  (traces
    ((recv (enc s r a ns (pubk v)))
      (recv (enc "cert" a i e (privk e)))
      (send (enc nv j v ns m (pubk s)))
      (recv
        (enc kp s jo m p
          (enc
            (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
            (invk i)) (pubk v)))
        (send (enc "valid" kp ns (pubk s))))
      ((send (enc "cert" a i e (privk e))))
      ((recv (enc r a k (pubk s))) (send (enc s r a ns (pubk v)))
        (recv (enc nv j v ns m (pubk s))) (send (enc nv j v m k)))
      ((recv (enc s nv j v m r (ltk a a)))
        (send
          (enc kp nv
            (enc kp s jo m p
              (enc
                (enc "hash" (enc "hash" a v r nv j jo hash) m p
                  hash) (invk i)) (pubk v)) (ltk a a))))
        ((send (cat c (enc r a k (pubk s)))) (recv (enc nv j v m k))
          (send (enc s nv j v m r (ltk a a)))
          (recv
            (enc kp nv
              (enc kp s jo m p
                (enc
                  (enc "hash" (enc "hash" a v r nv j jo hash) m p
                    hash) (invk i)) (pubk v)) (ltk a a))))

```

```

(send
  (enc kp s jo m p
    (enc
      (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
      (invk i)) (pubk v))))))
(label 29)
(parent 0)
(unrealized)
(shape)
(annotations ((0 1) v (says e (id a i))) ((0 2) v (ask r a j m))
  ((0 3) v (says a (meas i nv j jo m p)))
  ((0 4) v (approved r a nv)) ((1 0) e (id a i))
  ((2 1) s (verifier v))
  ((3 1) a (and (verifier v) (meas i nv j jo m p))))
(obligations
  ((0 1) v
    (implies (says e (id a i)) (says s (verifier v))
      (says e (id a i))))
  ((0 3) v
    (implies (ask r a j m) (says e (id a i))
      (says s (verifier v))
      (says a (and (verifier v) (meas i nv j jo m p)))
      (says a (meas i nv j jo m p))))))

```

A.2 Scenario

verifier



```

(defskeleton caves
  (vars (r m p j jo text) (ns nv data) (e s a v name) (kp skey)

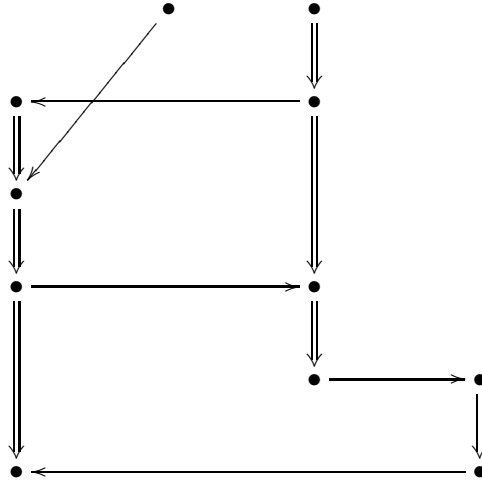
```

```

(hash i akey))
(defstrand verifier 4 (r r) (m m) (p p) (j j) (jo jo) (ns ns)
  (nv nv) (a a) (v v) (e e) (s s) (kp kp) (hash hash) (i i))
(non-orig (invk hash) (privk e) (privk s))
(uniq-orig nv)
(traces
  ((recv (enc s r a ns (pubk v)))
   (recv (enc "cert" a i e (privk e)))
   (send (enc nv j v ns m (pubk s)))
   (recv
    (enc kp s jo m p
      (enc
        (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
        (invk i)) (pubk v))))))
(label 61)
(unrealized (0 1) (0 3))
(comment "1 in cohort - 1 not yet seen"))

```

verifier epca server attester



```

(defskeleton caves
  (vars (r m p j jo r-0 text) (ns nv data) (e s a v a-0 s-0 name)
    (kp k kp-0 skey) (hash i akey))
  (defstrand verifier 4 (r r) (m m) (p p) (j j) (jo jo) (ns ns)

```

```

    (nv nv) (a a) (v v) (e e) (s s) (kp kp) (hash hash) (i i))
(defstrand epca 1 (a a) (e e) (i i))
(defstrand server 4 (r r-0) (m m) (j j) (ns ns) (nv nv) (a a-0)
  (v v) (s s) (k k))
(defstrand attester 2 (r r) (m m) (p p) (j j) (jo jo) (nv nv)
  (a a) (v v) (s s-0) (kp kp-0) (hash hash) (i i))
(precedes ((0 2) (2 2)) ((1 0) (0 1)) ((2 1) (0 0))
  ((2 3) (3 0)) ((3 1) (0 3)))
(non-orig (invk hash) (invk i) (privk e) (privk s))
(uniq-orig ns nv kp-0)
(operation nonce-test (added-strand server 4) nv (3 0)
  (enc nv j v ns m (pubk s)))
(traces
  ((recv (enc s r a ns (pubk v)))
    (recv (enc "cert" a i e (privk e)))
    (send (enc nv j v ns m (pubk s)))
    (recv
      (enc kp s jo m p
        (enc
          (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
          (invk i)) (pubk v))))
    ((send (enc "cert" a i e (privk e))))
    ((recv (enc r-0 a-0 k (pubk s)))
      (send (enc s r-0 a-0 ns (pubk v)))
      (recv (enc nv j v ns m (pubk s))) (send (enc nv j v m k)))
    ((recv (enc s-0 nv j v m r (ltk a a)))
      (send
        (enc kp-0 nv
          (enc kp-0 s-0 jo m p
            (enc
              (enc "hash" (enc "hash" a v r nv j jo hash) m p
                hash) (invk i)) (pubk v)) (ltk a a))))))
(label 65)
(parent 61)
(unrealized)
(shape)
(annotations ((0 1) v (says e (id a i))) ((0 2) v (ask r a j m))
  ((0 3) v (says a (meas i nv j jo m p))) ((1 0) e (id a i))
  ((2 1) s (verifier v))
  ((3 1) a (and (verifier v) (meas i nv j jo m p))))

```

```

(obligations
  ((0 1) v
    (implies (says e (id a i)) (says s (verifier v))
      (says e (id a i))))
  ((0 3) v
    (implies (ask r a j m) (says e (id a i))
      (says s (verifier v))
      (says a (and (verifier v) (meas i nv j jo m p)))
      (says a (meas i nv j jo m p))))))

```

A.3 Scenario

attester

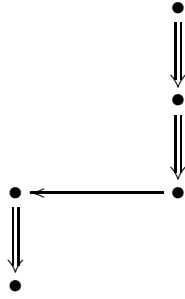


```

(defskeleton caves
  (vars (r m p j jo text) (nv data) (a v s name) (kp skey)
    (hash i akey))
  (defstrand attester 2 (r r) (m m) (p p) (j j) (jo jo) (nv nv)
    (a a) (v v) (s s) (kp kp) (hash hash) (i i))
  (non-orig (ltk a a) (invk hash) (privk v))
  (uniq-orig kp)
  (traces
    ((recv (enc s nv j v m r (ltk a a)))
      (send
        (enc kp nv
          (enc kp s jo m p
            (enc
              (enc "hash" (enc "hash" a v r nv j jo hash) m p
                hash) (invk i)) (pubk v)) (ltk a a))))))
  (label 66)
  (unrealized (0 0))
  (comment "1 in cohort - 1 not yet seen"))

```

attester client



```

(defskeleton caves
  (vars (r m p j jo text) (nv data) (a v s c name) (kp k skey)
    (hash i akey))
  (defstrand attester 2 (r r) (m m) (p p) (j j) (jo jo) (nv nv)
    (a a) (v v) (s s) (kp kp) (hash hash) (i i))
  (defstrand client 3 (r r) (m m) (j j) (nv nv) (c c) (a a) (v v)
    (s s) (k k))
  (precedes ((1 2) (0 0)))
  (non-orig (ltk a a) (invk hash) (privk v))
  (uniq-orig kp k)
  (operation encryption-test (added-strand client 3)
    (enc s nv j v m r (ltk a a)) (0 0))
  (traces
    ((recv (enc s nv j v m r (ltk a a)))
      (send
        (enc kp nv
          (enc kp s jo m p
            (enc
              (enc "hash" (enc "hash" a v r nv j jo hash) m p
                hash) (invk i)) (pubk v)) (ltk a a))))
      ((send (cat c (enc r a k (pubk s)))) (recv (enc nv j v m k))
        (send (enc s nv j v m r (ltk a a))))))
  (label 67)
  (parent 66)
  (unrealized)
  (shape)
  (annotations ((0 1) a (and (verifier v) (meas i nv j jo m p))))

```

(obligations))

A.4 Scenario

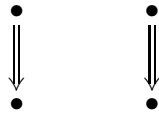
attester



```
(defskeleton caves
  (vars (r m p j jo text) (nv data) (a v s name) (kp skey)
    (hash i akey))
  (defstrand attester 2 (r r) (m m) (p p) (j j) (jo jo) (nv nv)
    (a a) (v v) (s s) (kp kp) (hash hash) (i i))
  (non-orig (invk hash) (privk v))
  (uniq-orig kp)
  (traces
    ((recv (enc s nv j v m r (ltk a a)))
      (send
        (enc kp nv
          (enc kp s jo m p
            (enc
              (enc "hash" (enc "hash" a v r nv j jo hash) m p
                hash) (invk i)) (pubk v)) (ltk a a))))))
  (label 68)
  (unrealized)
  (shape)
  (annotations ((0 1) a (and (verifier v) (meas i nv j jo m p))))
  (obligations))
```

A.5 Scenario

attester



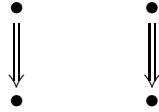

```

(defskeleton caves
  (vars (jo r m p j text) (nv data) (a v s name) (kp skey)
    (hash i akey))
  (defstrand attester 2 (r r) (m m) (p p) (j j) (jo jo) (nv nv)
    (a a) (v v) (s s) (kp kp) (hash hash) (i i))
  (deflistener jo)
  (non-orig (invk hash) (privk v))
  (uniq-orig jo kp)
  (traces
    ((recv (enc s nv j v m r (ltk a a)))
      (send
        (enc kp nv
          (enc kp s jo m p
            (enc
              (enc "hash" (enc "hash" a v r nv j jo hash) m p
                hash) (invk i)) (pubk v)) (ltk a a))))
      ((recv jo) (send jo))))
  (label 69)
  (unrealized (1 0)))

```

A.6 Scenario

attester



```

(defskeleton caves
  (vars (p r m j jo text) (nv data) (a v s name) (kp skey)
    (hash i akey))
  (defstrand attester 2 (r r) (m m) (p p) (j j) (jo jo) (nv nv)
    (a a) (v v) (s s) (kp kp) (hash hash) (i i))
  (deflistener p)
  (non-orig (invk hash) (privk v))
  (uniq-orig p kp)
  (traces
    ((recv (enc s nv j v m r (ltk a a)))
      (send

```

```

(enc kp nv
  (enc kp s jo m p
    (enc
      (enc "hash" (enc "hash" a v r nv j jo hash) m p
        hash) (invk i)) (pubk v)) (ltk a a))))
((recv p) (send p)))
(label 71)
(unrealized (1 0)))

```

A.7 Scenario

server



```

(defskeleton caves
  (vars (b mesg) (r m j d text) (ns nv data) (v s a name)
    (k kp skey))
  (defstrand server 8 (b b) (r r) (m m) (j j) (d d) (ns ns)
    (nv nv) (a a) (v v) (s s) (k k) (kp kp))
  (non-orig (privk v) (privk s))
  (uniq-orig ns)

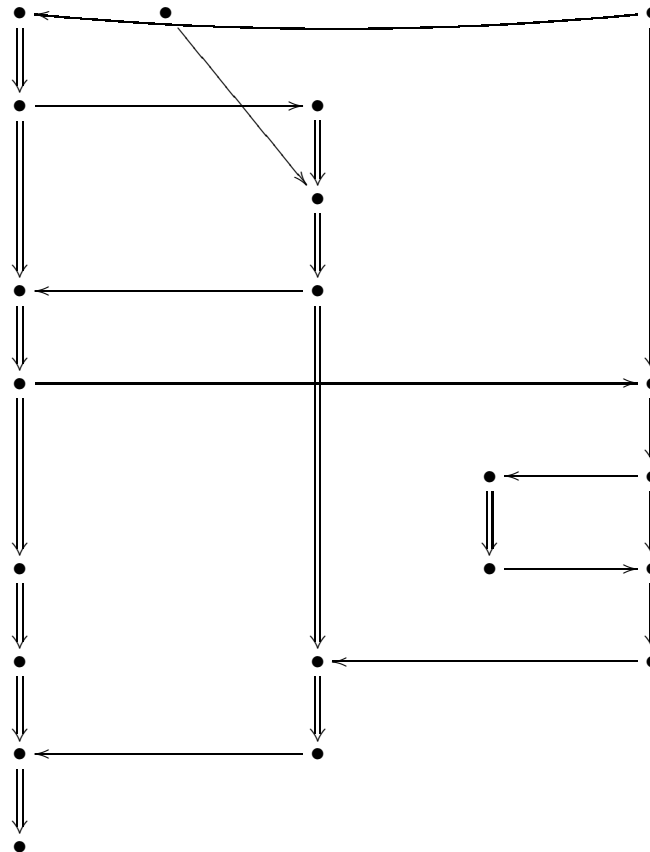
```

```

(traces
  ((recv (enc r a k (pubk s))) (send (enc s r a ns (pubk v)))
   (recv (enc nv j v ns m (pubk s))) (send (enc nv j v m k))
   (recv b) (send b) (recv (enc "valid" kp ns (pubk s)))
   (send (enc "data" d kp))))
(label 73)
(unrealized (0 2) (0 6))
(comment "1 in cohort - 1 not yet seen"))

```

server epca verifier attester client



```

(defskeleton caves
  (vars (b msg) (d m j p jo r text) (ns nv data) (v a e c s name)
        (k kp skey) (hash i akey))

```

```

(defstrand server 8 (b b) (r r) (m m) (j j) (d d) (ns ns)
  (nv nv) (a a) (v v) (s s) (k k) (kp kp))
(defstrand epca 1 (a a) (e e) (i i))
(defstrand verifier 5 (r r) (m m) (p p) (j j) (jo jo) (ns ns)
  (nv nv) (a a) (v v) (e e) (s s) (kp kp) (hash hash) (i i))
(defstrand attester 2 (r r) (m m) (p p) (j j) (jo jo) (nv nv)
  (a a) (v v) (s s) (kp kp) (hash hash) (i i))
(defstrand client 5
  (b
    (enc kp s jo m p
      (enc (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
        (invk i)) (pubk v))) (r r) (m m) (j j) (nv nv) (c c)
    (a a) (v v) (s s) (k k) (kp kp))
  (precedes ((0 1) (2 0)) ((0 3) (4 1)) ((1 0) (2 1))
    ((2 2) (0 2)) ((2 4) (0 6)) ((3 1) (4 3)) ((4 0) (0 0))
    ((4 2) (3 0)) ((4 4) (2 3)))
  (non-orig (ltk a a) (invk hash) (invk i) (privk v) (privk e)
    (privk s))
  (uniq-orig ns nv k kp)
  (operation nonce-test (contracted (kp-0 kp)) ns (0 6)
    (enc "valid" kp ns (pubk s)) (enc nv j v ns m (pubk s))
    (enc s r a ns (pubk v)))
  (traces
    ((recv (enc r a k (pubk s))) (send (enc s r a ns (pubk v)))
      (recv (enc nv j v ns m (pubk s))) (send (enc nv j v m k))
      (recv b) (send b) (recv (enc "valid" kp ns (pubk s)))
      (send (enc "data" d kp)))
    ((send (enc "cert" a i e (privk e))))
    ((recv (enc s r a ns (pubk v)))
      (recv (enc "cert" a i e (privk e)))
      (send (enc nv j v ns m (pubk s)))
      (recv
        (enc kp s jo m p
          (enc
            (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
            (invk i)) (pubk v)))
        (send (enc "valid" kp ns (pubk s))))
    ((recv (enc s nv j v m r (ltk a a)))
      (send
        (enc kp nv

```

```

(enc kp s jo m p
  (enc
    (enc "hash" (enc "hash" a v r nv j jo hash) m p
      hash) (invk i)) (pubk v)) (ltk a a)))
((send (cat c (enc r a k (pubk s)))) (recv (enc nv j v m k))
  (send (enc s nv j v m r (ltk a a)))
  (recv
    (enc kp nv
      (enc kp s jo m p
        (enc
          (enc "hash" (enc "hash" a v r nv j jo hash) m p
            hash) (invk i)) (pubk v)) (ltk a a)))
    (send
      (enc kp s jo m p
        (enc
          (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
            (invk i)) (pubk v))))))
(label 96)
(parent 73)
(unrealized)
(shape)
(annotations ((0 1) s (verifier v))
  ((0 6) s (says v (approved r a nv)))
  ((0 7) s (and (approved r a nv) (resource r d)))
  ((1 0) e (id a i)) ((2 1) v (says e (id a i)))
  ((2 2) v (ask r a j m))
  ((2 3) v (says a (meas i nv j jo m p)))
  ((2 4) v (approved r a nv))
  ((3 1) a (and (verifier v) (meas i nv j jo m p))))
(obligations
  ((0 6) s
    (implies (verifier v) (says e (id a i))
      (says v (ask r a j m)) (says v (approved r a nv))
      (says a (and (verifier v) (meas i nv j jo m p)))
      (says v (approved r a nv))))
  ((2 1) v
    (implies (says s (verifier v)) (says e (id a i))
      (says e (id a i))))
  ((2 3) v
    (implies (says s (verifier v)) (says e (id a i))

```

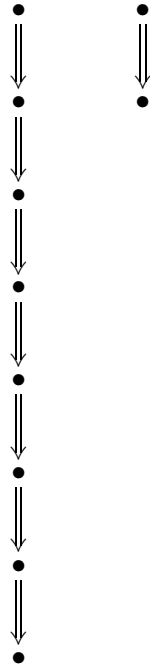
```

(ask r a j m)
(says a (and (verifier v) (meas i nv j jo m p)))
(says a (meas i nv j jo m p))))))

```

A.8 Scenario

server



```

(defskeleton caves
  (vars (b mesg) (d r m j text) (ns nv data) (v s a name)
    (k kp skey))
  (defstrand server 8 (b b) (r r) (m m) (j j) (d d) (ns ns)
    (nv nv) (a a) (v v) (s s) (k k) (kp kp))
  (deflistener d)
  (non-orig (privk v) (privk s))
  (uniq-orig d ns)
  (traces
    ((recv (enc r a k (pubk s))) (send (enc s r a ns (pubk v)))
      (recv (enc nv j v ns m (pubk s))) (send (enc nv j v m k))
      (recv b) (send b) (recv (enc "valid" kp ns (pubk s)))

```

```

      (send (enc "data" d kp))) ((recv d) (send d)))
(label 107)
(unrealized (0 2) (0 6) (1 0)))

```

A.9 Scenario

client

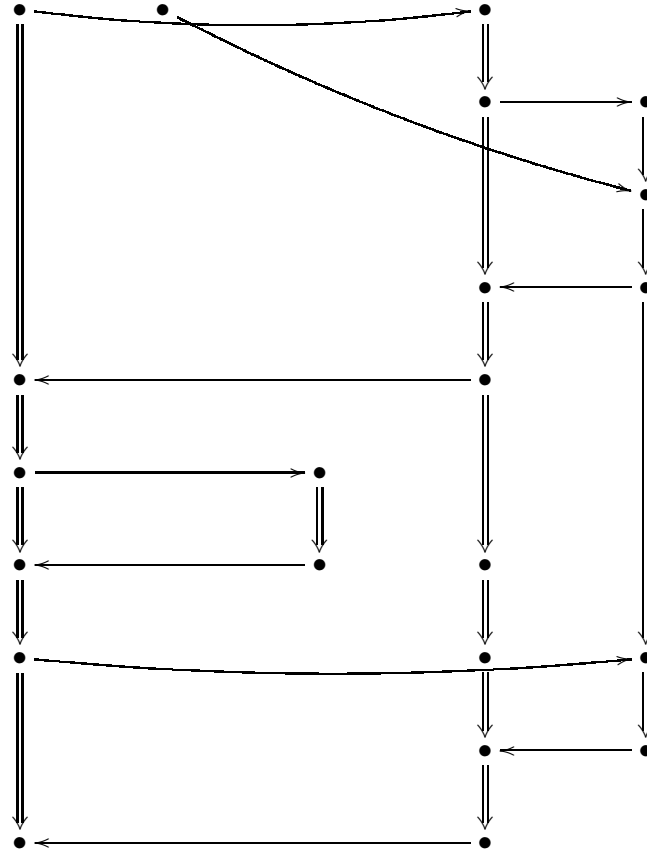


```

(defskeleton caves
  (vars (b msg) (r m j d text) (nv data) (a v s c name)
        (k kp skey))
  (defstrand client 6 (b b) (r r) (m m) (j j) (d d) (nv nv) (c c)
    (a a) (v v) (s s) (k k) (kp kp))
  (non-orig (ltk a a) (privk v) (privk s))
  (uniq-orig k)
  (traces
    ((send (cat c (enc r a k (pubk s)))) (recv (enc nv j v m k))
      (send (enc s nv j v m r (ltk a a)))
      (recv (enc kp nv b (ltk a a))) (send b)
      (recv (enc "data" d kp))))
  (label 157)
  (unrealized (0 1) (0 3))
  (comment "2 in cohort - 2 not yet seen"))

```

client epca attester server verifier



```
(defskelton caves
  (vars (b msg) (d p jo r m j text) (nv ns data) (s c a v e name)
    (kp k skey) (hash i akey))
  (defstrand client 6
    (b
      (enc kp s jo m p
        (enc (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
          (invk i)) (pubk v))) (r r) (m m) (j j) (d d) (nv nv)
      (c c) (a a) (v v) (s s) (k k) (kp kp))
    (defstrand epca 1 (a a) (e e) (i i))
    (defstrand attester 2 (r r) (m m) (p p) (j j) (jo jo) (nv nv)
      (a a) (v v) (s s) (kp kp) (hash hash) (i i))
```



```

(defstrand server 8 (b b) (r r) (m m) (j j) (d d) (ns ns)
  (nv nv) (a a) (v v) (s s) (k k) (kp kp))
(defstrand verifier 5 (r r) (m m) (p p) (j j) (jo jo) (ns ns)
  (nv nv) (a a) (v v) (e e) (s s) (kp kp) (hash hash) (i i))


```

```

(enc
  (enc "hash" (enc "hash" a v r nv j jo hash) m p
    hash) (invk i)) (pubk v)) (ltk a a))))
((recv (enc r a k (pubk s))) (send (enc s r a ns (pubk v)))
  (recv (enc nv j v ns m (pubk s))) (send (enc nv j v m k))
  (recv b) (send b) (recv (enc "valid" kp ns (pubk s)))
  (send (enc "data" d kp)))
((recv (enc s r a ns (pubk v)))
  (recv (enc "cert" a i e (privk e)))
  (send (enc nv j v ns m (pubk s)))
  (recv
    (enc kp s jo m p
      (enc
        (enc "hash" (enc "hash" a v r nv j jo hash) m p hash)
        (invk i)) (pubk v)))
    (send (enc "valid" kp ns (pubk s)))))
(label 210)
(parent 157)
(unrealized)
(shape)
(annotations ((0 5) c (says s (resource r d)))
  ((1 0) e (id a i))
  ((2 1) a (and (verifier v) (meas i nv j jo m p)))
  ((3 1) s (verifier v)) ((3 6) s (says v (approved r a nv)))
  ((3 7) s (and (approved r a nv) (resource r d)))
  ((4 1) v (says e (id a i))) ((4 2) v (ask r a j m))
  ((4 3) v (says a (meas i nv j jo m p)))
  ((4 4) v (approved r a nv)))
(obligations
  ((0 5) c
    (implies (says e (id a i))
      (says a (and (verifier v) (meas i nv j jo m p)))
      (says s (verifier v))
      (says s (and (approved r a nv) (resource r d)))
      (says v (ask r a j m)) (says v (approved r a nv))
      (says s (resource r d))))
    ((3 6) s
      (implies (says e (id a i))
        (says a (and (verifier v) (meas i nv j jo m p)))
        (verifier v) (says v (ask r a j m))

```

```

      (says v (approved r a nv)) (says v (approved r a nv))))
((4 1) v
  (implies (says e (id a i)) (says s (verifier v))
    (says e (id a i))))
((4 3) v
  (implies (says e (id a i))
    (says a (and (verifier v) (meas i nv j jo m p)))
    (says s (verifier v)) (ask r a j m)
    (says a (meas i nv j jo m p))))))

```